

LIBRARY
OF THE
MASSACHUSETTS INSTITUTE
OF TECHNOLOGY

HD28
.M414
no.762-75



MASS. INST. TECH.
1975

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Alfred P. Sloan School of Management

Center for Information Systems Research

A HIERARCHICAL APPROACH TO INFORMATION SYSTEM DESIGN

by

John Donovan and Henry Jacoby*

REPORT CISR-5

SLOAN WP 762-75

January 21, 1975

Center for Information Systems Research
Massachusetts Institute of Technology
Alfred P Sloan School of Management
50 Memorial Drive
Cambridge, Massachusetts, 02139

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Alfred P. Sloan School of Management

Center for Information Systems Research

A HIERARCHICAL APPROACH TO INFORMATION SYSTEM DESIGN

by

John Donovan and Henry Jacoby*

REPORT CISR-5

SLOAN WP 762-75

January 21, 1975

* We would like to acknowledge the contributions throught this book of Grant Smith and Lou Gutentag, Smith most especially for his conceptual contributions to the DML and DDL, and Gutentag for his application of this work to NEEMIS.

HD28
.M414
no762-15

A HIERARCHICAL APPROACH TO INFORMATION SYSTEM DESIGN

by

John J. Donovan and Henry D. Jacoby

OUTLINE

ABSTRACT

PART I

1. Structure of Paper
2. Characterization of Data Problems
3. Characteristics of Management Information Systems Problems
4. Hierarchical Approach

PART II

1. Level 1: Machine Instruction
2. Level 2: The Operating System
3. Level 3: File Systems (Table Facility)
4. Level 4: Relational Operator Level
 - 4.1 Relational History
 - 4.2 Relational Model
 - 4.3 Definition of Operators
 - 4.3.1 Set Operators as Applied to Relations
 - 4.3.2 Relational Operators
 - 4.3.2.1 Example Using Operators
 - 4.3.3 Implementation Operators
 - 4.3.4 New Operators

0723237

A HIERARCHICAL APPROACH TO INFORMATION SYSTEM DESIGN

BY

John Donovan and Henry Jacoby*

ABSTRACT

This paper presents an approach to the development of management information systems that is particularly applicable to systems with the following characteristics:

- several classes of users, each of which has a different degree of sophistication
- complex and changing security requirements
- data exhibits complex and changing inter-relationships
- needs of the information system changing
- must be built quickly and inexpensively
- complex data validation requirements

The approach is hierarchical in that the functional tasks of the system are grouped and ordered such that each group depends only on functions of the group beneath it. Each group is called a level. We maintain that not only does this approach facilitate the implementation of management information systems to fulfill the above needs but provides a sound theoretical basis for investigating properties of completeness, integrity, correctness, and performance. Some of these theoretical approaches are also presented. We also maintain that such hierarchical approaches, in general, provide a practical way of decomposing complex system designs into manageable implementation schemes.

We feel that some of the primitives of the levels described here will eventually be placed into the hardware architecture of machines. We have applied this approach to the development of information systems for public policy decisions regarding energy for the states, New England Energy Management Information System (NEEMIS).

5. Level 5: Security Validation and Performance

6. Level 6: DDL/DML

 6.1 An Example of Using DDL

 6.2 Syntax Specification of DDL

 6.3 Implementation of the DDL

 6.4 The Data Manipulation Language (DML)

 6.5 Example of Use of DML in Accessing NEEMIS

 6.6 Complete Syntax Specification of DML

 6.7 Implementation of DML

7. Performance

8. Further Issues and Investigations in Hierarchies

9. Further Investigations in Relations

10. Comparison of Other Views of Data

11. Summary

PART 1

1. Structure of Paper

The paper is divided into two parts. The first exposes the key problems in representing data in a computer system and gives an overview of the hierarchical approach. The second gives detailed discussion of each level of a hierarchy for information systems.

The levels we present are: bare machine, operating systems, store and retrieval of relations, relational operators, security and validation, a Data Definition Language (DDL) and a Data Manipulation Language (DML), report generator and user interface, modeling facility. We formalize some levels, give the present state of research at MIT of each level, and present future theorems and potential fruitful research directions of each level. The store and retrieval relations are further divided into multiple sub-levels.

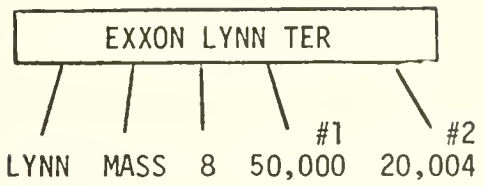
2. Characterization of Data Problems

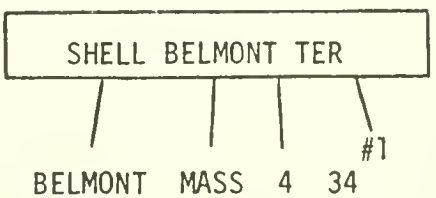
The basis of an information system is data. Let us address ourselves to the problems of data complexity and complications. Figure 1 depicts an example of two data series typical of those that economists are accustomed to handling. One could perform all sorts of statistical operations on the inventory series -- regressions, averages, standard deviations, etc.

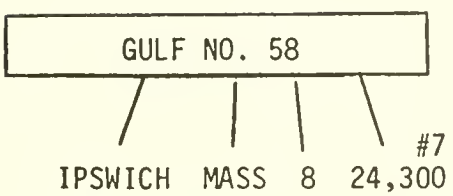
However, for policymaking it is important to store more complex information, such as the relations between data. For example, Figure 2 depicts four data items: the names of terminals, their address, region number, and inventory value of different types of fuel. Let us further

complicate Figure 2 by representing data in relation to the owners, suppliers, and all terminals in region 8. Figure 3 depicts these data items and their relationships. Now visualize what such a diagram would look like for all terminals in the U.S. and all possible relationships -- a mess! The basic problems are:

- How can such information be represented logically?
- How should an implementor view such data?
- What operators should exist to manipulate such data?
- What mechanisms should be available to validate it?
- How can its protection be ensured?
- All of the above within the constraints of:
 - good performance (low operating cost of system)
 - recognition of the fact that all the relationships might change; the types of data series available might change







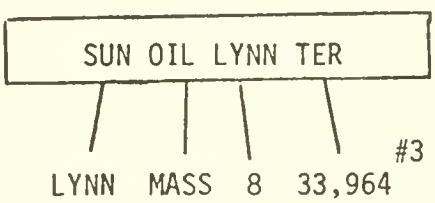


Figure 2

Terminal Data

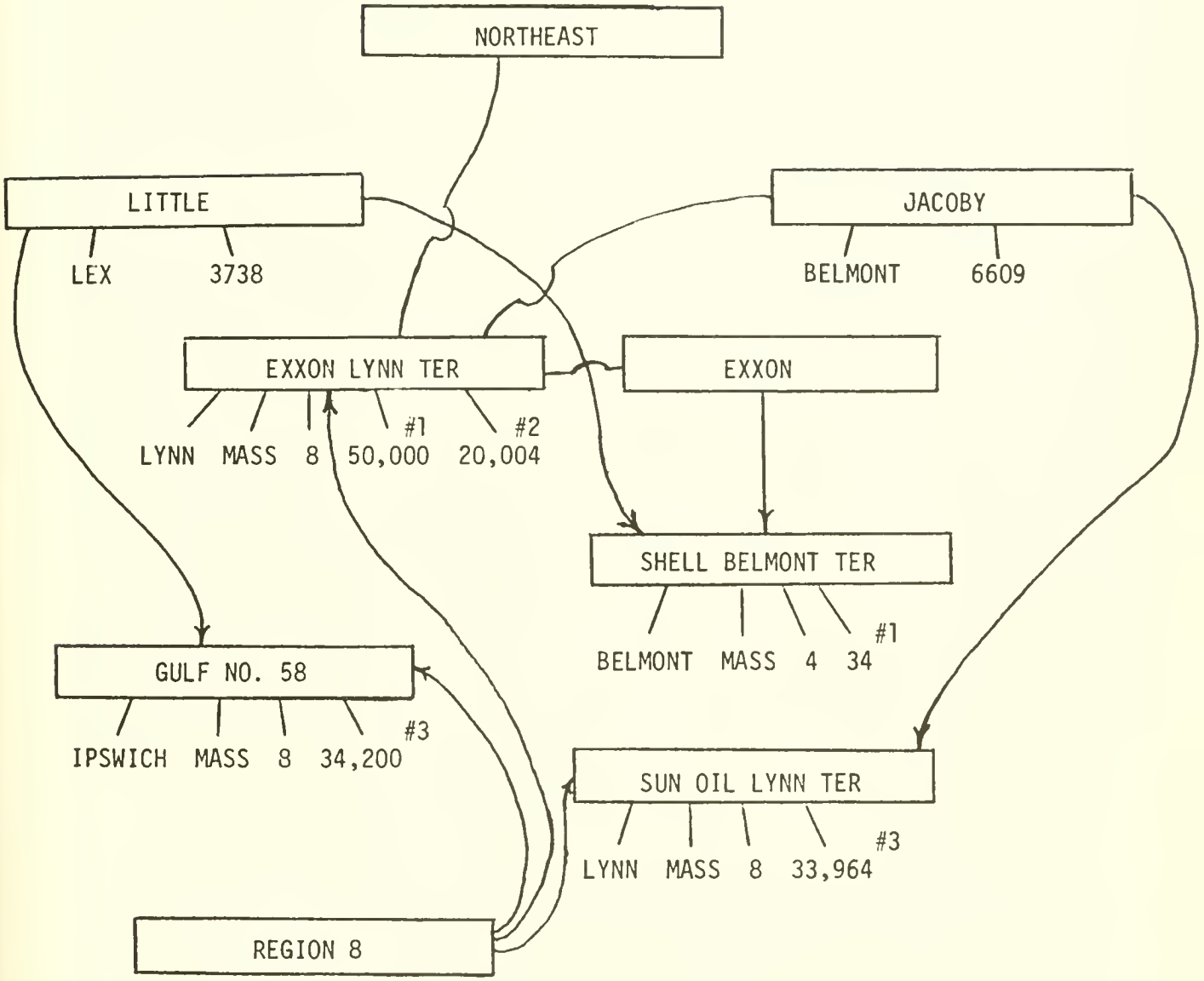


Figure 3

Complex Data

3. Characteristics of Management Information Systems Problems

What is so complicated about an information system? The following lists some of those problems.

- Representation of data
- Storage
- Retrieval
- Manipulation of Data
- Use as Base for Models for Public Policy
- Satisfy different degree of sophistication of users

All of the above problems msut be addressed within the constraints of:

- low cost
- good performance
- users will want changes as the evolution of the user and needs of a system change
- levels of users

4. Hierarchical Approach

The basic idea behind this approach is to divide a problem into groups of functions (levels). Each group is ordered such that it depends only on the group below it. Let us take a simple example to develop an intuitive feel for this approach.

Suppose a carpenter wishes to build a house. One view he may have of his basic components would be as is depicted in Figure 4. That is, he views all his basic components as wood, nails, putty, glass, etc.

A hierarchical approach is depicted in Figure 5 where the carpenter views his components in levels. The second level windows are composed of components only from the levels below. Doors may be composed of windows and any of the components of the inner level. This carpenter has simplified the construction task and also may have simplified the "debugging" task. Namely, if a door is not operating correctly and if the levels below it are debugged, then the fault must be in the door. (Note: recursion is not allowed. Windows cannot have door assemblers in them).

Data Hierarchical Model

The motivation here is to choose a logical representation of data that is divorced from the physical implementation.

Historical:

File blocking is an example of an early technique for giving the user of a computer a different view of data than what appears in the physical implementation. (An example of blocking is having several logical records placed into one physical record on a tape.)

The paramount advantage of giving a user a logical representation that is independent and separate from the physical implementation is that the physical representation may be changed without affecting the application programmer. This separation is an example of a two-level hierarchy. One of the first explanations of this hierarchical concept and its application

to file systems appears in [Madnick, 1970], in SYSTEMS PROGRAMMING [Donovan, 1972] and later in OPERATING SYSTEMS [Madnick & Donovan, 1974]. The most notable early expositions of this hierarchical concept appears in "T.H.E. Multiprogramming System", [Dijkstra, 1968].

Extending this two-level heirarchy we find the hierarchy depicted in Figure 6 is applicable to the design, implementation, and study of information systems.

A user viewing the bare machine sees instructions like "load, add, store, multiply". A user viewing level 2 sees instructions like "Get more memory", "Give me a device". A user at level 3 may store and retrieve tables. A user at the operator level has three operators on tables. For example, an operator would be "Find all common entries between tables". Auser at the data security level can only access tables under prescribed rules. A user at the DML and DDL level can use a cryptic English for query information stored in tables or use the DDL to define new tables. A user at level 8 may activate packages. A user at level 9 may create and activate models.

Note that each level may be viewed as the user of the level below it, as it uses the primitives of levels below it. For example, a request at level 8 to "Select all terminals in Lynn, Mass. of over 50,000 gallons of #1 fuel would activate the data security level to check the access rights of such a request. The data security level would use the operators of level 4 on securirty tables, which contain information as to the protection rights on the terminal table. These same operators will be used on the / terminal table to

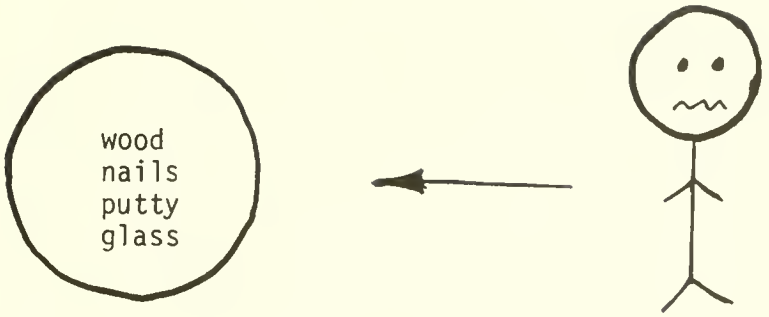


Figure 4

Non-hierarchical View from Carpenter

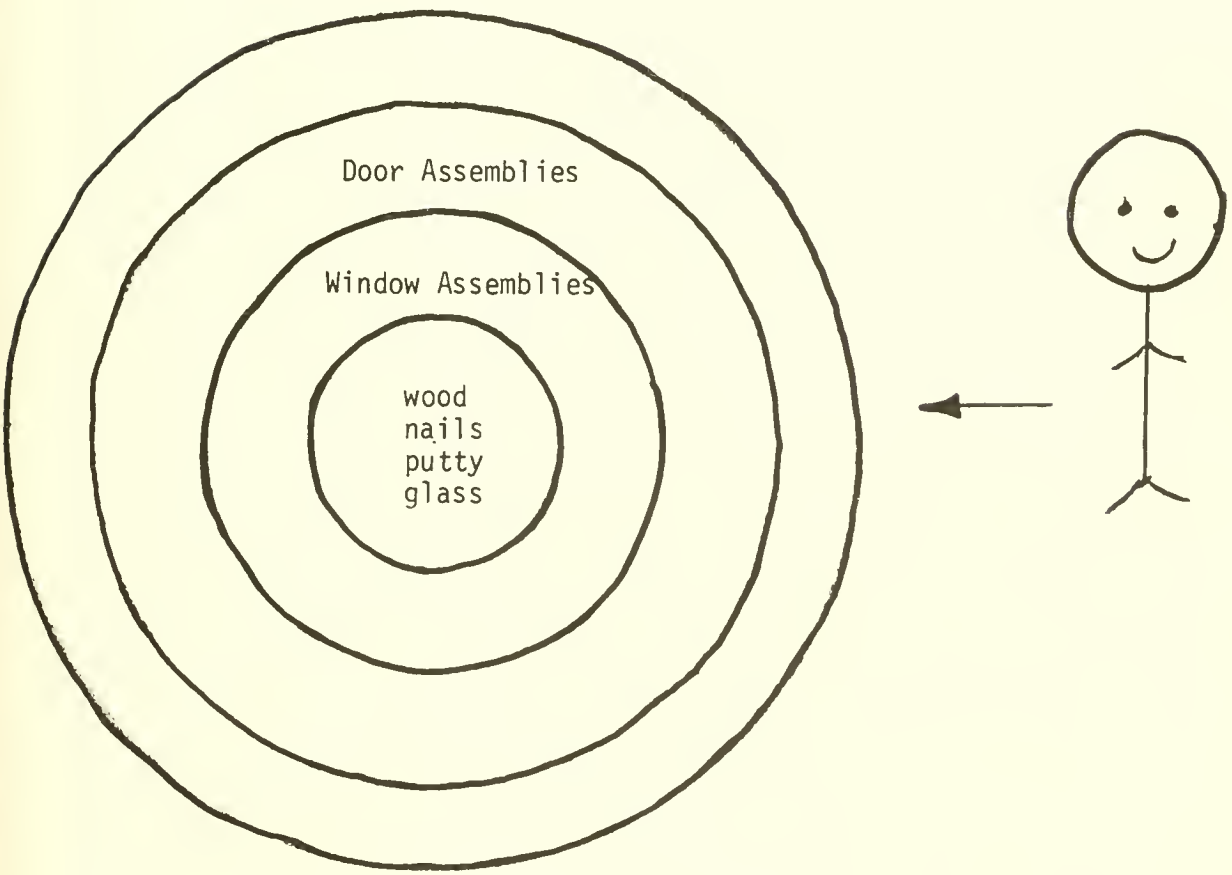


Figure 5

Hierarchical View

get the desired information. The request is carried further down the hierarchy until the actual machine instructions and I/O commands of the bare machine are initiated to satisfy the request.

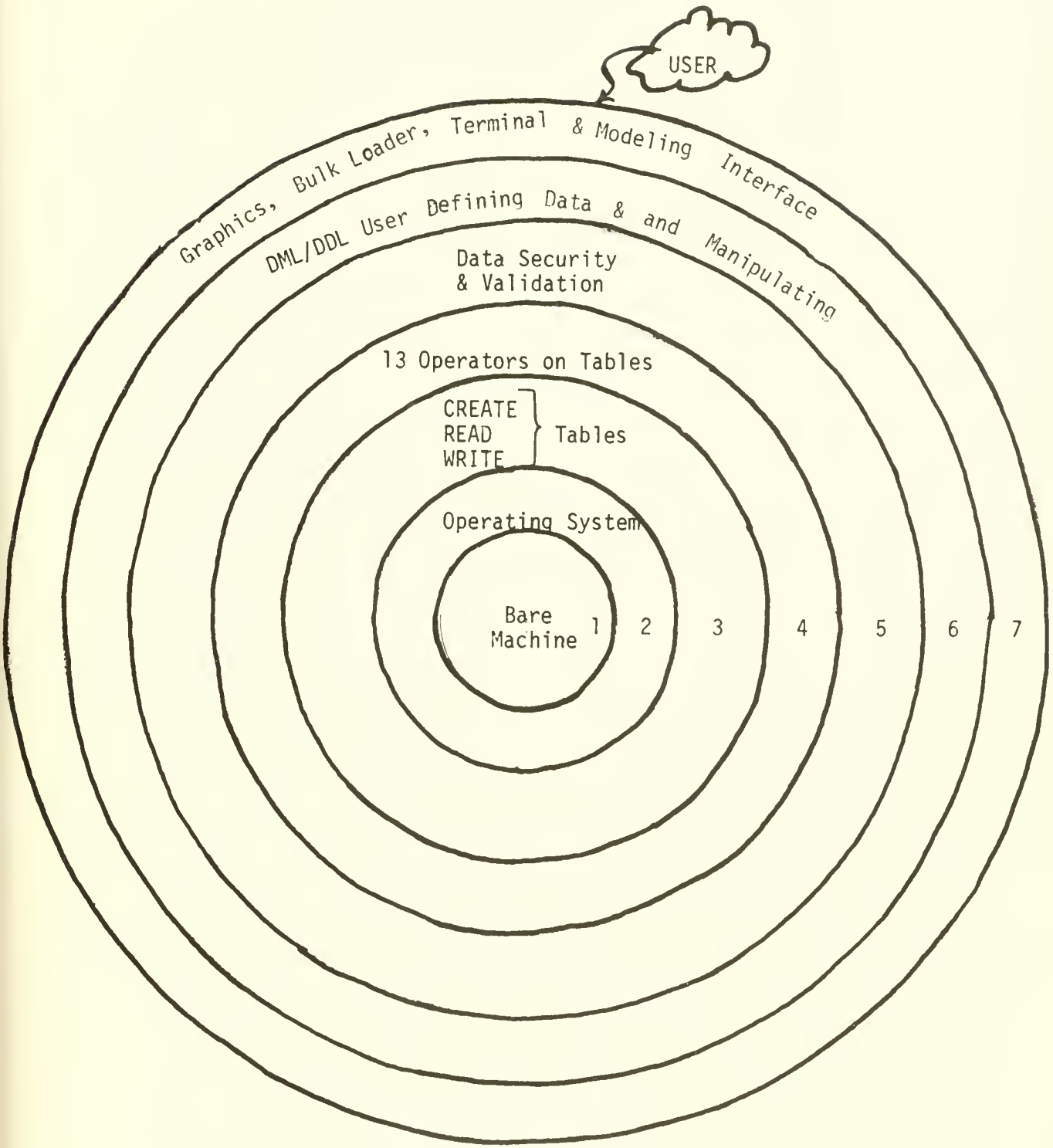


Figure 6

Hierarchical Implementation

PART II

The basic idea behind a hierarchical implementation is that each level consists of algorithms that depend only on (call) algorithms in levels beneath it. No level is allowed to call itself, therefore, recursion on a level is not allowed.

1. Level 1: Machine Instruction.

Level 1 presents the user with machine instruction.

2. Level 2: The Operating System.

A user of this level sees instructions such as "Give me a device", "Give me more memory". The operating system is a resource manager in that it manages the resources of the computer, memory, CPU time, and devices. We further divide the operating system into sublevels [OPERATING SYSTEMS, Madnick & Donovan, 1974]. For this paper we will assume that either an operating system exists or the reader will refer to the references if he should have to build one.

3. Level 3: File Systems (Table Facility)

The function of this level is to store and retrieve by symbolic name tables. A user at this level sees instructions such as CREATE TABLE, READ TABLE, WRITE TABLE. Some / standard file systems, for example, IMS, MULTICS, VSAM, etc., have such a facility (simply by equating a table to a file). To build one from start, we advocate that this level be subdivided into the sublevels of Figure 7.

Let us take an example and briefly describe the functions of each sublevel.

READ TABLE TERMINAL

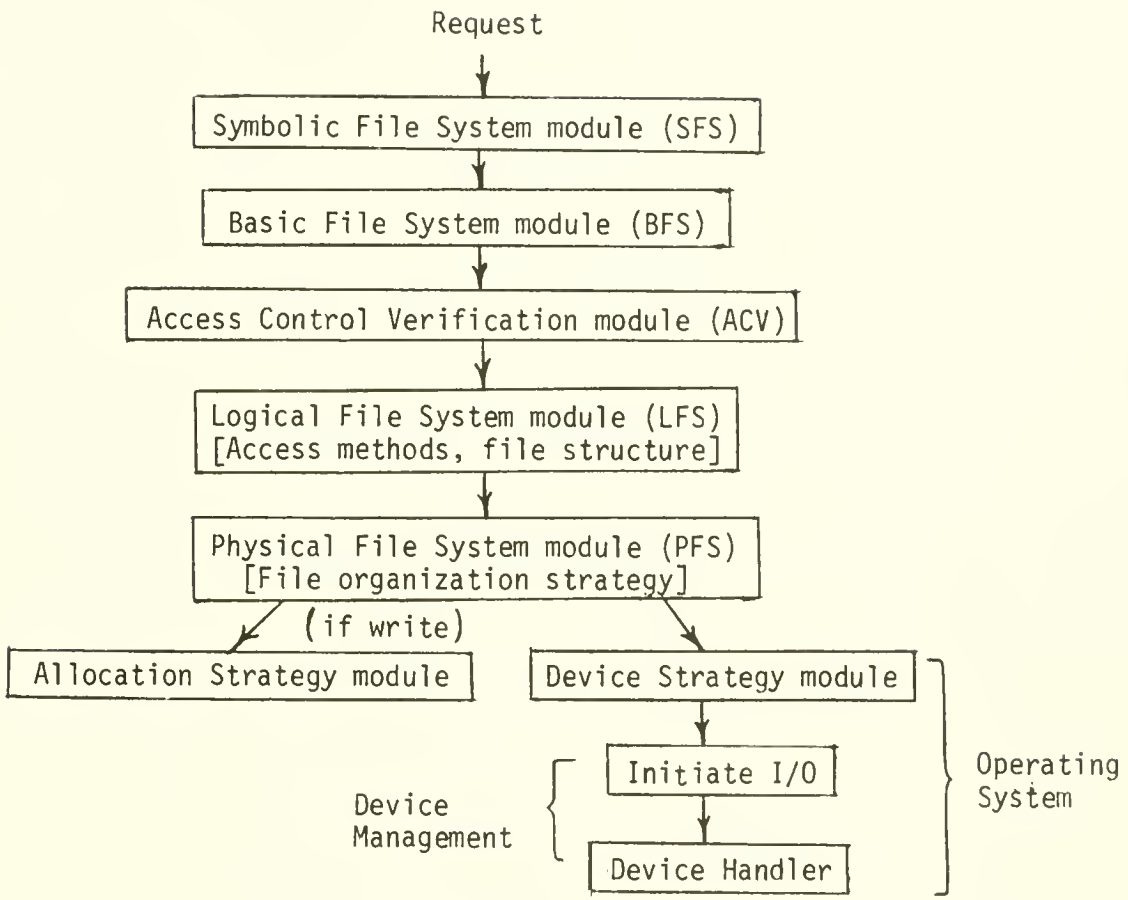


Figure 7

Hierarchical Model of a File System

The Symbolic File System's function is to map the symbolic reference into a unqiue identifier. This sublevel may read a master directory to find the unique id of 'TERMINAL'. If recursion were allowed, the SFS (i.e., find the unique id of the master directory) level would call itself to READ master directory/. Since recursion is not allowed, SFS cannot call itself to find the unique id of the master directory. Thus the master directory's unique id must be known by the SFS, and to read this directory, the level below is called passing the unique id.

We call the process of identifiying what each level must know to prevent recursion "unwinding the recursion".

When the Basic File System is called, it is given the unique ID of the table requested. The function of the BFS is to find all the information about the table (e.g., its size, location, access, etc.).

The function of access control verification is to provide the mechanism for allowing sharing of tables by multiple users with different access rights.

After access is checked, then the Logical File System is called. The LFS is concerned with mapping the structure of the logical records onto the linear byte string view of a file.

The primary function of the Physical File System is to perform the mapping of the Logical Byte Address into Physical Block Addresses. For example, the table to be read might be physically scattered in several different parts of a disk. The LFS might calculate its logical address as logical bytes 4000 to 5000 where the PFS would compute the specific track and cylinder numbers for each component.

The Allocation Strategy Module is only activated on a write (or create) request when more space is needed for a table or a new table is created. This module finds the space.

The Device Strategy Module, I/O initiator, and device handler are actually levels of the operating system. They create channel programs, request I/O, schedule the I/O, and handle interrupts.

4. Level 4: Relational Operator Level

This hierarchical approach for the design of an information system does not require that all of the levels below a certain point be implemented in a hierarchical manner, though we believe that even for the low levels just discussed this approach is best.

At level 4 we have implemented the thirteen operators on tables (some times called relations). The operators are: cartesian product, union, intersection, projection, diadic restriction, monadic restriction, join, composition, permutation, compuation, difference, inversion, and ordering.

4.1 Relational History

Relational representation can be looked upon as variants of Post-canonical systems [Post: 1943], Church's logical systems [Church: 1941], or somewhat more recently, Smullyan's elementary formal systems [Smullyan: 1961], and most recently, Donovan's canonic systems [Donovan: 1967].

Codd, however, is recognized as the first to apply this sort of logical system to the representation of data [Codd: 1971]. Another information algebra was proposed by R. Bosak [Bosak: 1962].

We know of six attempts to implement on a computer data in relational form: ISG [Smith: 1973], MACAIMS [Goldstein, Strnad: 1971], SEQUEL [Chamberlin: 1974], COLARD [Bracchi: 1972], RIL [Fehher:1972], and M.I.T.'s

RDMS. Most of these implementations are nearly functionally equivalent to our DDL and DML of the next level.

The only practical application of a relational system (that we know of) is to an energy information system for aiding public policy decisions in New England [Donovan & Jacoby: 1974]. We have further reported in that paper an extension of these concepts to include protection and validation mechanisms.

Sloan's contributions to date lie in the area of the first application, ex-tensions of concepts to handle protection validation, additional operators, hierarchical implementation, performance considerations, and a Data Manipulation and Data Definition Language.

4.2 Relational Model

A user at this level views data as relations. A relation in its simplest form is a tabel where each column is a domain and each row is an entry. Let us formalize this concept and define some operators on relations.

The cartesian product of two sets, A and B, is denoted by A x B, and is defined by:

$$A \times B = \left\{ (a,b):\ a \in A\ ,\ b \in B \right\}.$$

The expanded cartesian product, X, of n sets B_1, B_2,..., B_n is defined by:

$$S(B_1,B_2,\ldots,B_n) = \left\{ (b_1,b_2,\ldots,b_n):\ b_j \in B_j \text{ for } j = 1,2,\ldots,n \right\}.$$

The elements of such a set are called n-tuples, or just tuples . When $n = 1$, $X(B_1) = B_1$ since no distinction is made between a 1-tuple and its only component.

Suppose $b = (b_1, b_2, \ldots b_m)$ and $c = (c_1, c_2, \ldots, c_n)$. The concatenation of b with c is the (m + n)-tuple defined by

$$b \| c = (b_1, b_2, \ldots b_m, c_1, c_2, \ldots e_n).$$

R is a relation on the sets $(B_1, B_2, \ldots, B_n)$ if it is a subset of $X(B_1, B_2, \ldots, B_n)$. A relation is accordingly a special kind of set. Its members are all n-tuples where n is a constant called the degree of the relation. Relations of degree 1 are called unary, degree 2 binary, degree 3 ternary, degree n n-ary. The sets on which a relation is defined are called its underlying domains. For data base purposes, we (unlike the earlier work of Codd and others) are concerned with data consisting of many types, e.g., integers, characters, floating point, pointers, binary, boolean, etc.

Note the elements (n-tuples) of a relation have no implied order, thus insertion or deletion operators are simplified.

4.3 Definition of Operators

If data is represented in this relational model, what are the appropriate operators?

All operators create a new relation either from a single relation or from two relations. Using the mnemonic "diadic" to refer to operators that operate on two relations and "monadic" for operators that operate on one, we may divide the logical operators into four categories:

Category	Operator	Symbol	Type
Traditional set operators as applied to relations	Union	U	Diadic
	Intersection	N	Diadic
	Cartesian Product	X	Diadic
Relational operators appearing in literature	Projection	P	Monadic
	Join	*	Diadic
	Composition	·	Diadic
	Permutation	M	Monadic
	Restriction	R	Both
Implementation operators	Inversion	I	Monadic
	Ordering	π	Monadic
New operators defined in our system	Compaction	C	Monadic
	Difference	—	Diadic

Notation:

R_i is the name of the ith relation

c_i = cardinality (# members) of the ith relation

n_i = degree of ith relation

d_{ij} = jth domain of R_i,; j=1,...n_i

$v_m(d_{ij})$ = mth value of d_{ij} ; m=1,...c_i

t_i = n_i-tuple in relation R_i

i.e.: $t_i \in \{v_\alpha(d_{i1}), v_\alpha(d_{i2}),...v_\alpha(d_{ini})\}$

$L(\alpha)$ = length of list $\alpha \in \{1,...c_i\}$

$\forall\alpha$ = "for all values of" α

$\emptyset$ = null set (i.e.: $R_i = \emptyset \Rightarrow c_i = 0$}

$\alpha \subseteq \beta$ = α is a subset of β

$\alpha \subset \beta$ = α is a proper subset of β (i.e.: $\alpha \neq \beta$)

The following examples will be used throughout to explain definitions:

R1 =

(name	soc_sec_#	phone	dept_#)
(SMITH	213-97-1666	232-1500	15)
(MACAVOY	621-49-2990	356-5175	15)
(JACOBY	413-00-0029	484-7352	6)
(SMITH	839-41-6942	253-0410	6)

R2 =

(name	soc_sec_#	phone	dept_#)
(MADNICK	217-61-7232	253-6671	15)
(SMITH	213-07-1666	232-1500	15)
(MACAVOY	621-49-2990	356-5175	15)

R3 =

(person	age	city)
(MADNICK	31	PEABODY)
(MACAVOY	34	IPSWICH)
(SMITH	23	BOSTON)

R4 =

(name	phone)
(SMITH	232-1500)
(MACAVOY	356-5175)

R10 =

(person	age	city	street #)
(MADNICK	31	PEABODY	18)
(MACAVOY	34	IPSWICH	43)

Example: R6 = R1 N R2; results in:

R6 =	(SMITH , 213-07-1666 , 232-1500 , 15)
	(MACAVOY, 621-49 2990 , 356-5175 , 15)

Cartesian Product. Suppose you wanted to form a new relation from two relations where each element of the new relation consisted of every possible parsing of the elements of the existing relations. The cartesian product 'X' would perform this task.

$$R_i = R_j \times R_k \qquad (j = k \text{ is valid}).$$

Note: if $n_j > 1$ or $n_k > 1$, then each t_j (or t_k) msut be treated as a single domain so that effectively $n_j = n_k = 1$.

$$n_i = n_j + n_k$$

$$c_i = c_j * c_k$$

$$R_i = \{(v_\alpha(d_{j1}), v_\beta(d_{k1})),\ \alpha = 1,\ldots c_k\ ;\ \alpha = 1,\ldots c_j\}$$

i.e., R_i = {ordered pairs with first member from R_j, and second member from R_k}

Example: R5 = R4 x R4 ; results in:

R5 =	((SMITH , 232-1500),(SMITH , 232-1500))
	((SMITH , 232-1500),(MACAVOY , 356-5175))
	((MACAVOY , 356-5175),(SMITH , 232-1500))
	((MACAVOY , 356-5175),(MACAVOY , 356-5175))

4.3.1 Set Operators as Applied to Relations

Union: Suppose you would like to create a new relation that consisted of all the elements of two other relations without redundant entries. The union of two relations would perform this task. Using the symbol U we may formally define union as

$R_i = R_j \cup R_k$; ($j = k$ is valid).

$c_i = c_j + c_k - c_{(R_j \cap R_k)}$ i.e., duplicates deleted automatically

$n_i = \max(n_j, n_k)$

$R_i = \{t_i : t_i \in R_j \underline{OR}\ t_i \in R_x\}$

Example: R = R1 U R2; results in

R5 =

(SMITH	213-07-1666	232-1500	15)
(MACAVOY	621-49-2990	356-5175	15)
(JACOBY	413-00-0029	484-7352	6)
(SMITH	839-41-6942	253-0410	6)
(MADNICK	217-61-7232	253-6671	15)

$C_{R5} = 5$; $n_{R5} = 4$

Intersection: Suppose you wish to create a new relation whose elements consisted of only the common elelemnts of two other relations. The interseciton of those two relations would perfomr this task. Using the symbol N we may formally define the intersection operator.

$R_i = R_j \cap R_k$; ($i = j = k$ is valid)

(note that if $n_j \neq n_k$, then $R_i = \emptyset$)

$R_i = \{t_i : t_j = t_k\}$ Duplicate t_i are removed

$n_i = n_k = n_j$

4.3.2 Relational Operators [Codd: 1972] *

Projection. Suppose you wish to create a new relation that consisted of only some of the domains of an existing relation. Projection could do this task. (Projection has its equivalence in propositional calculus, the existential qualifier [Church: 1943]. Formally, the projecttion 'P' is defined

$R_i = R_j \; P \; (d_{j\ell}), \; \ell \subseteq \{1,2,\ldots n_j\}$

$n_i = L(\ell)$

$c_i = c_j$ (Note: redundant entries not automatically removed -- use the "compaction" operator for this purpose)

$R_i = \{d_{j\ell} : \ell \subseteq \{1,2,\ldots n_j\}\}$

Example: R5 = R2 P (name, phone); results in:

(MADNICK , 253-6671)
(SMITH , 232-1500)
(MACAVOY , 356-5175)

Join. Suppose you wished to create a new relation from two existing relations such that each element of the new relation was the concatenation of elements of the existing ones. Further, you only wanted to concatenate those elements whose domains had certain properties. The joining of these two relations would do it. Formally, the join '*' is defined (we find the following definition more natural than Codd's):

$R_i = R_j(d_{j\ell}) * R_k((\Theta, d_{km}))$;

$\Theta ::= > \mid < \mid = \mid \neg\Theta$

$\ell \in \{1,2,\ldots n_j\}$

$m \in \{1,2,\ldots n_k\}$

and $d_{j\ell}$ and d_{km} must be of the same data type (i.e., must be joinable).

*Note: We have somewhat changed some of the definitions for implementation and use reasons. E.g., we never automatically eliminate duplicate rows in Projection; we do eliminate one of the duplicate columns in Join.

$n_i = n_j + n_k - 1$ (no duplication of join domain)

$c_i = c_j + c_k - c(v_\alpha(d_{j\ell}) = v_\beta(d_{km}),\ \alpha = 1,\ldots c_j;\ \beta = 1,\ldots c_m)$

$$R_i = \{d_{j\beta},\ d_{k\alpha}\ (\beta = 1,\ldots n_j\ ;\ \alpha = 2,\ldots n_k) : V_\gamma(d_{j\ell})\,\Theta\, V_\delta(d_{km}),\ \gamma = 1,\ldots c_j\ ;\ \delta = 1,\ldots c_k\}$$

Example (1): R6 = R2 (name) * R3 (=, person); results in:

	(soc_sec_#	phone	dept_#	name	age	city)
R6 =	{(217-61-7232	253-6671	15	MADNICK	31	PEABODY),
	(213-07-1666	232-1500	15	SMITH	23	BOSTON),
	(621-49-2990	356-5175	15	MACAVOY	34	IPSWICH)}

Example (2): R6 = R3 (city) * R4(>, name); results in:

	(soc_sec_#	age	city	phone)
R6 =	{(MADNICK	31	PEABODY	617-1400)}

(Note that conceptually this example does not make sense; it simply illustrates the use of * when Θ is not "=".)

Composition. Suppose you wanted to do a join operator but you did not want the duplicate column to appear at all. You would perform a composition. Composition '.' is formally defined as:

$$R_i = R_j\ (d_{j\ell}) \cdot R_k\ (d_{km})$$

$\ell \in \{1,\ldots n_j\}$

$m \in \{1,\ldots n_k\}$

$d_{j\ell}$ & d_{km} must be joinable (of the same data type).

$$n_i = n_j + n_k - 2$$

$$c_i = c_j + c_k - c(v_\alpha(d_{j\ell}) = v_\beta(d_{km}),\ \alpha = 1,\ldots c_j,\ \beta = 1,\ldots c_k)$$

$$R_i = [R_j(d_{j\ell}) * R_k(d_{km})]\ P\ (d_{j\beta})$$

$$\beta = 1,\ldots d_{j(\ell-1)},\ d_{j(\ell+1)},\ldots n_j$$

(i.e., remove domain d_j on which R_j and R_k were joined)

Example: R5 = R2 (name) · R3 (person) ; results in:

	(soc_sec_#	phone	dept_#	age	city)
R5 = {(217-61-7232	253-6671	15	31	PEABODY)	
	(213-07-1666	232-1500	15	23	BOSTON),
	(621-49-2990	356-5175	15	34	IPSWICH)}

Permutation. Suppose you wish to interchange some domains of a relation. Permutation 'M' performs this task and is formally defined:

$$R_i = R_j\ M\ (d_{j\ell})\ ;\quad \ell \overset{\text{ordered}}{\subseteq} \{1,\ldots n_j\}\ (\ell) = n_j.$$

$$n_i = n_j\ ;\ c_i = c_j\ ;$$

The only effect of this operator is to reorder the domains in a relation.

Example: R5 = R3 M (person, city, age); results in:

	(name	city	age)
R5 = {(MADNICK	PEABODY	31),	
	(MACAVOY	IPSWICH	34),
	(SMITH	BOSTON	23)}

Diadic Restriction. Perhaps the most common task you might perform would be to ask for all the elements in a relation A that have anything to do with the elements of a relation B. This is a restriction of A by B and is formally defined, using the symbol |, as:

$$R_i = R_j\ (d_j\ell)\ |\ R_k\ ((\Theta_m,\ d_{km}));\ \ell \subseteq \{1,\ldots n_j\}$$
$$m \subseteq \{1,\ldots n_k\}$$

where: $L(\ell) = L(m)$

and $n_k \quad n_j \leq n_j$

then: $n_i = n_j$

Θ_m :: = > | < | = | $\neg\Theta$

$$R_i = \{t_j : v\ (d_{j\ell})\ \Theta_m\ v_\beta(d_{km})\ ;\ \ell,\ m = 1,\ldots L(\ell);\ \alpha = 1,\ldots c_j\ ;\ \beta = 1,\ldots\}$$

Example (1):

R6 = R2 (name , phone) | R4 ((=, name) , (=, phone)); results in:

R6 = {(SMITH 213-07-1666 232-1500 15),
(MACAVOY 621-49-2990 356-5175 15)}

Example (2):

R6 = R2 (phone) | R4 ((>, phone)); results in:

R6 = {(MADNICK 217-61-7232 253-6671 15,
(MACAVOY 621-49-2990 356-5175 15}

(Note: t_1 of R6 appears because 253-6671 > 232-1500. The fact that 253-6671 < 356-5175 does not affect this.)

Monadic Restriction. Suppose you wish to have all the elements of a single relation that conformed to a condition on a domain. The monadic restriction of that relation would do that and is formally defined as:

$$R_i\ R_j\ (d_{j\ell}) \mid ((\Theta_m, d_{jm}));$$

$$\ell \subseteq \{1,\ldots n_j\}$$

$$m \subseteq \{1,\ldots n_k\}$$

$$L(\ell) = L(m)$$

$$\Theta_m :: = > \mid < \mid = \mid \neg \Theta_m$$

$$n_i = n_j$$

$$R_i = \{t_j : v_\alpha(d_{j\ell})\ \Theta_m\ v_\beta(d_{jm})\ ;\ \ell,m = 1,\ldots L(\ell);\ \alpha,\ \beta = 1,\ldots c_j\}$$

Example: R6 = R10 (age) | ((<, street_#)); results in:

R6 = {(MACAVOY, 34, IPSWICH, 43)}.

4.3.2.1 Example Using Operators

Let us take an example demonstrating the use of this view and a series of these operators to satisfy a query. Referring to Figure 0.3, we may construct a possible representation of this data by the following five realtions:

TERMINAL (TERNAME, CITY, STATE, REGION)

INVENTORY (TERNAME, FUELTYPE, INVEN, CONFIDEN)

OWNER (OWNERNAME, ADDRESS, PHONE)

TERMINALOWNER (TERNAME, OWNERNAME)

SUPPLIERS (SUPNAME, TERNAME, DISTRIBUTOR...)

Note that if we had included inventory information in the terminal relation, there would have been many empty fields since all fuel types are not present at most terminals.

Let us now use the basic operators of diatic restriction and projection on the above relations to extract the information needed to answer the following question.

"WHAT IS TELEPHONE NUMBER FOR OWNER OF GULF NO. 48 TERMINAL?"

The following expression when evaluted gives the result:

$$\{ \text{OWNER} \mid \{ \text{TERMINALOWNER} \mid \text{'gulf no 48 [=, TER NAME]} \} \text{P(OWNERNAME)} \} \text{P(phone)}$$

Starting with the innermost parenthesis, the expression is evaluated. By restricting the TERMINALOWNER relation with the instance "gulf no. 48", the resulting relation is then projected on the OWNERNAME domain, which results in a realtion of all desired owners. This relation is used to restrict the owner relation to produce a new relation containing all information about the desired owners. This result is then projected on the phone domain to get the desired phone numbers.

Note that we never were concerned with the physical storage of this data, nor were we concerned about the number of entries of the results. The operators do not depend on the number of data elements, their order, or their type.

4.3.3 Implementation Operators

These are two operators that we have implemented in the system. We call them implementation operators as they are not necessary in the theoretical relational calculus, but when used effectively, they can greatly assist performance of the practical computer implementation.

Inversion Operator:

Often a relation will be accessed by queries on specific domains or a combination of domains. The data base management system would then maintain additional information -- conceptually indices for the specified domains -- to facilitate efficient handling of frequent requests. These domains are specified by the user as 'keys' or inversions and are maintained by the system.

We call this an implementation operation in that in the relational calculus we may access any domain, and similarly any implementation of this model could access any element by specifying the values of domains. This general access could only be accomplished by a linear search of each element, as elements are not ordered. However, an inverted domain could employ more efficient search techniques.

Ordering Operator:

Another implementation operator that we are implementing (see future memo of Grant Smith) is the ordering operator. As the relational mathematics do not use the inversion operator the mathematics assumes all entries in a relation are unordered. However, often for reports or for selection it is desirable to order either a relation's entries or its domains.

4.3.4. New Operators

We find the following two operators to be helpful in operating on data at the relational level. These operators were defined and implemented in the Sloan system [Smith: 1974].

Difference. Suppose you wished the inverse of restriction. That is, suppose you wished to obtain all the elements of a relation A that have nothing to do with the elements of a realtion C. The difference of A and B (A-B) gives a relation composed of such elements. Formally,

$R_i = R_j - R_k$;

(Note that if $n_j \neq n_k$ then: $n_i = \max(n_j, n_k)$,

$c_i = c_j$

$R_i = R_j$.)

$n_i = n_j = n_k$

$c_i = c_j - c(R_j \; N \; R_k)$

$R_i = \{t_i : t_i \in R_j \text{ and } t_i \notin R_k\}$

Example: R6 = R1 - R2; results in:

R6 = {(GRANGER, 413-00-0029, 536-5176, 6),

(SMITH, 839-41-6942, 253-0410, 6)}

Compaction. If you took the projection of a relation it is possible for you to get a new relation with multiple entries of the same element. If you then wished to eliminate these duplicate elements you could perform a compaction. Formally, a compaction 'C' is defined:

$R_i = C(R_j)$; (i = j is valid)

$n_i = n_j$

$R_i = \{t_i : t_i \neq t_\alpha \quad \forall_\alpha \neq i\}$ [OR: $R_i = R_j \; N \; R_j$]

5. Level 5: Security Validation and Performance

By using relations about relations this level extends the relational model to provide security, validation, and performance information. That is, all relations in the system have associated with them protection, validation and perfomrance information that is kept in another relation. or all requests this level checks (by use of relation operators of the level below acting on the associated protection relation) the access rights of all relations before any access is made. It also examines the validation information associated with a relation before any inserts are made into that relation. Lastly, it updates perfomrance information associated with all relations before any access is made to them.

In our present implementation, access control is applied to two general aspects of the system:

1) The structure of the system

2) The contents (data of the system).

The types of access that can be specified for system structure control are:

a) read -- user may see system descriptors

b) delete -- user may delete parts of the system

c) modify -- may change existing structure, but not delete it

d) insert -- user may define new relations, but not alter existing ones

e) owner -- user created relation, and so can do anything with it, including giving other people access rights to it, or denying himself certain rights

f) trap -- this invokes a monitoring program to oversee any actions the user may perform

The facility exists within the current implementation of this level add an to / additional 12 controls, without making changes to existing structures. These controls may be applied to relations as a whole or as individual domains.

6. Level 6: DDL/DML

This level / presents to the user a Data Definition and a Data Manipulation Language. A Data Definition Language (DDL) allows a user to specify the structure and form of the data base. The DDL will accept this specification and will produce an appropriate relational data base system. The DDL also provides a facility for loading bulk data into the newly constructed relational system. Such loading of bulk data would be either from punched cards, from magnetic tapes, or from a computer magnetic disk file.

The Data Manipulation Language (DML) is a language that allows a user to query any data series stored in a relational system. An internal M.I.T. document, "The Internal Intermediate Language" dated April 29, 1974 describes a complete DDL and DML that have been specified at M.I.T.. This work is further evolving out of research by Smith, Madnick, and Donovan. That document is an updated version of a DDL and DML specified in November, 1974 (Smith). For the NEEMIS facility in 1975 we will deliver a working and

debugged subset of the DDL and DML described in the referenced document. The following section describes that subset.

6.1 An Example of Using DDL

With most information management systems, the design of the system -- that is, the design of the data base -- is a vital step in the operation. If done incorrectly, it is often impossible, and usually extremely costly (in dollars and man years) to restructure the data base to more ably suit the needs.

Not so with NEEMIS (the New England Energy Management Information System) [Donovan & Jacoby: 1974].

A sample session of the DDL might be:

```
system:     ENTER COMMAND:
user:       define domains
            name character, soc_sec_# numeric 9,
            zip character, age num 3, address char;
system:     ENTER COMMAND:
user:       create relation
            employee (name, soc_sec_#, age) (primary key 1),
            employee_data (name, address, zip) (pk 1, required 2);
system:     RELATIONS DEFINED
            ENTER COMMAND
user:       stop.
```

This session would establish the two relations and permit data to be entered immediately.

Given this simplicity and flexibility, redefining the data base ceases to be a major task.

Facilities of the Definition Sublanguage

The user first defines the domains he will use in some relation, as well as the type of data that will appear in that domain. (The digit following "numeric" specifies the maximum number of digits that can appear in a value of the domain.) The system makes use of the data type information when the user enters data -- it can automatically check to see that the value being entered matches what the domain expects (e.g., 19B is not a valid number).

Once the domains are defined, the user defines the relations. Definition of a relation consists of:

a) a name for the relation

b) the domains of that relation

c) options.

The options are:

1) Primary key (or "pk"), which specifies which domains are to be used as the primary key

2) Required (or "req") tells the system that certain domains must have values when the data is entered, otherwise it will not be accepted by the system. (Primary key domains are required by default). For example, in "employee_data" the system will not accept data unless it includes a name, and an address.

6.2 Syntax Specification of DDL

(using BNF, see Donovan, SYSTEMS PROGRAMMING)

The following table defines/all the commands and options of the DDL which will be implemented by November, 1975. For further details and examples of this subset of DDL, refer to Section 2 of the previously referenced document by Smith.

In the table, the * indicates that the feature in question is implemented in the prototype NEEMIS system.

Definition Statements

*DEFINE DOMAIN[S]

data specifiable for domains:

1) * <domain name> :: = any character string ≤ 40 characters

2) <data type> :: = CHAR[ACTER]* | NUM[ERIC] <range> * |

BIT { (<size>) | (<info>) } | FLOAT[ING] / INT[EGER] <range> |

INTERNAL | CHOICE (<value>,...)

2.1) <range> :: = <lower>, <upper> | <upper>*
2.2) <size> :: = <positive integer>

3) IN <relation name>

4) P[ERMIT] <TYPE> ACCESS [TO] { <user name> / (<user name>,...)

<type> :: = <access> | (<access>,...)
<access> :: = any of 19 modes

Note 1: (3) is mutually exclusive with (2)
Note 2: for BIT data, the (<info>) is used to name bits in the field -- e.g., DEFINE DOMAIN ACCESS_WORD BIT (READ, WRITE...); READ is bit 1, etc.

*DEFINE RELATION

1) *<name>(<domain>,...)[(<options>)]

1.1) *<name> :: = any character string ≤ 40 characters

1.2) (<domain name> :: = any predefined domain name or existing relation <name>.TID

1.3) <options> :: = <option >,< options> | <option >

<option> :: = *{PRIMARY KEY / PK} (<domain>[,...]) |

*REQ[UIRED](<domain>[,...]) |

*INV[ERT](<domain>[,...])

2) P[ERMIT]<type> ACCESS [TO] ...same as DEFINE DOMAIN

*DROP {DOMAIN (<domain>,...) IN {(<relation>,...) / ALL} / RELATION <relation name>}

DEFINE INVERSTION[S] IN <relation> (<domain name>,...)

SET {MONITOR / DEBUG / ERRORMSG} {ON / OFF}

6.3 Implementation of the DDL

Again, using the hierarchcal approach, this level is simply and relatively straightwardly implemented using the operators of the level below. A DROP command causes an entry in the master relation to be eliminated. The operators of the relational operator level are used to find this entry in the master relation directory. The master directory is a relation containing an entry for each relation created by the DDL. The domains of the master relation describe the domains of the relations created.

6.4. The Data Manipulation Language (DML)

This language allows the user to access data stored in a relational system. The entire DML is described in section 3 of the previously referenced Smith document.

In the subset of DML that will be operational by November, 1975 we will include five very powerful commands.

i) DISPLAY which retrieves specified data and prints it on the desired device (e.g., the console).

ii) DELETE deletes specified entries or relations from the data base

iii) GET INTO retrieves specified data and places it into a specified file. This command provides the mechanisms for transferring data stored in the relational data base to a modelling system such as TROLL.

iv) INSERT places data into a specified entry into the data base.

v) UPDATE changes data in the data base

6.5 Example of Use of DML In Accessing NEEMIS

The following are sample queries against a data base which contains the tables:

TERMINAL (TERMINALID, NAME, CITY, STATE, ZIP CODE, AFFILIATION)

SUPPLY CAPACITY (TERMINALID, FUELTYPE, FUELAMT, DATA)

SUPPLIER (SUPPLIERNO, NAME, VOLUME, FUELTYPE, DISTNO)

DISTRIBUTORS (DISTNO, NAME, ADDRESS, CITY, STATE, INVENTORY, FUELTYPE)

Question 1

```
DISPLAY TERMINAL (NAME, ADDRESS, CITY)
        FOR STATE = 'MASSACHUSETTS',
```

This question causes the listing of the name, address and city of all terminals in the state of Massachusetts.

Question 2

```
DISPLAY TERMINAL (NAME, CITY) FOR FUELAMT>1000, FUELTYPE = 'GASOLINE',
                                 CITY = 'LYNN', DATE = 'NOV74'
```

This lists the name and address of all terminals in Lynn which have over 1000 gallons of gasoline on hand in November, 1974.

6.6 Complete Syntax Specification of DML

The complete syntax of these commands are given below. Included are all options and arguments that will be available as of November, 1975. The * next to an item indicates that this facility exists in the present NEEMIS protype.

```
{ DISPLAY*
  DELETE
  GET INTO <file name>(<format info>) { <domain>[,...]*
                                      { ALL

                          FOR { <domain> = '<value>'[,...]*
                              { ALL                        ;

INSERT* { IN <relation>* { FROM <file name>(<format info>);
        {                { <domain> = '<value>'[,...]*    ;
        {                { '<value>', '<value>', '<value>' ;
        {
        { <domain>='<value>' [,...];

UPDATE { <domain name> TO { EXCLUDE
       {                  { INCLUDE ('<value>'[,...])
       {                  { '<new value>' FOR { ALL
       {                  {                   { <domain> = '<val>',...
       {
       { ENTRY IN relation TO { (<domain> = '<vlaue>',...)
       {                      { ('<value>',...)
                                                FOR { ALL
                                                    { <domain>= '<val>',...
```

(The { XCLUDE / INCLUE } ('<value>'[,...]) is primarily for BIT data -- such as access control words. It does not reset any other bits -- only the one(s) affected.)

e.g.: UPDATE ACCESS_WORD to XCLUDE ('READ') FOR USER_NAME = 'JONES', DOMAIN = 'SALARY';

ALL commands are terminated by a semi-colon -- ';'.)

(NOTE: The syntax is continually being revised and the above represents the DDL, DML as of November 1975.)

6.7 Implementation of DML

The DML consists of a parser that recognizes the key words and translates the request into the appropriate sequence of operation of the operator level. For example, the two DML commands

DISPLAY TERMOWNER.OWNERNAME FOR TERNAME = GULF 48

DISPLAY OWNER.PHONE FOR OWNERNAME = ' ' result of first command.

is translated into the oeprator sequence given in the example of the previous level.

Level 7:

At level 7 we have implemented a bulk loader that allows loading of data from cards into a relation. We have written vairous graphic query languages and facilities. We have interface the system to virtually any temrinal. At this level we have implemetned an enter command that allows a user to access operators at lower levels.

7. Performance

At each level of the hierarchy the amount of time to execute the programs and the amount of storage necessary for these programs is of concern.

At the time of writing we have done very little analysis of performance of our implementation or of our application to energy. IBM has recently (effective November 1974) signed a two-year joint study agreement with M.I.T.'s Sloan School, a major part of which is to investigate performance issues, specifically, a comparison and analysis of Sloan School's implementation and IBM's implementation of a relational data model as well as Sloan's application to energy. Further, ARPA is considering a proposal that will be foramlly submitted in 1975 addressing the extensions of our approach to very large data bases.

Performance is an important issue of each level of the hierarchical model of data.

Basically, for our performance work we employ two approaches:

(1) Analytical approach

(2) Empirical approach

For our analytical research we will develop models of each level of the hierarchy of Figure 6. For our empirical work at each level we will define controlled experiments and take imperative measurements. The purpose of the empirical data is twofold:

1) To verify our analytic models. We want to be assured that we have considered all influential factors in our formulation of models.
2) To initialize / paramaters of models. Some models will involve parameters that must be initialized. Values of these parameters will be determined empirically.

Example:

An example of this dual approach is our analysis of the paging performance within Level 2.

In OPERATING SYSTEMS [Madnick and Donovan: 1974] and in Madnick's Ph.D. thesis [Madnick: 1973] we built a model of paging and proved that there exists cyclic program references that can cause page fetch frequency to increase significantly if the page size used is decreased (e.g., reduced by half). Furthermore, the proof of the theorem below shows that the limit to this increase is a linear function of primary storage size (the more memory, the worse performance can be!!)

THEOREM

For any two demand-fetch LRU-removal two-level storage systems, S and S', with page sizes N and N'=N/2 and primary store sizes $|M^1|$ and $|M^1|'=2\ M]$, respectively, there exists a cyclic page trace, P = (Pc)*, where $Pc = 2(M^1 + 1)$, such that the steady-state page fetch frequency ratio, /r/, equals $M] + 1$.

PROOF: See Madnick [Madnick: 1973].

Further, to minimize the bad effect of this anomaly and yet gain the potential good effects of reducing page size, we presented an algorithm that Madnick calls tuple coupling. This algorithm can be added to any existing paging algorithm and limit the bad effects of reducing page size.

THEOREM:

For any two demand-fetch two-level storage systems, S and S', with page sizes N and N'=N/2, respectively, the use of the "tuple-coupling" approach for S' in conjunction with a removal algorithm that is "tuple-couple-able" is sufficient to guarantee that the page fetch frequency ratio, r, cannot exceed the value 2 for all possible page traces, P.

On the empirical side, two people independently [Hatfield: 1972; Seligman: 1968] have performed experiments that verified that the theoretical phenomenon not only occurs but can occur frequently. Hatfield performed studies in the hardware environment of the IBM 360 Model 67 with programs running under CP-67/CMS. Seligman observed the same siutation in a cache system with much smaller page sizes. In a more recent work [Donovan and Madnick: 1975] we developed a probabalistic model of Level 1 for analysis of data security and privacy of this level.

Possible Models of Level 3:

Level 3 is the file system that is concerned with the physical representation of data. One possible physical organization of a series of data is in a tree as in Figure 3.1. (For example, organize one level of nodes by a person's last name, the next level by first name). Questions that come to mind immediately are: What would be the number of levels and nodes

to minimize the number of accesses? What is the minimum access path to a particular node? What are the best searching algorithms? A possib le analytical approach to the second question may be taken by using a topographical representation of the tree of Figure 3.1 and assigning truth values to each link and device functions to minimize the paths to a given node.

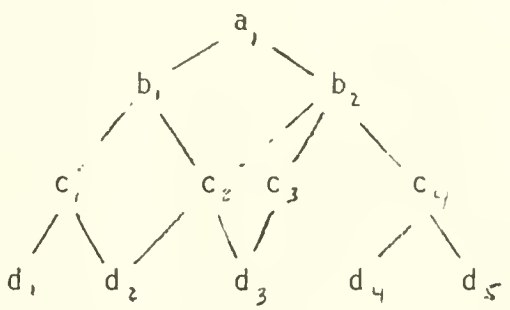


Figure 3.1
Tree Structure as a Physical Representation

The first question of what is the minimum number of nodes can be addressed using analyses similar to those of Severence's. Another way of expressing this question is to find a compromise between the number of comparisons performed in the branching nodes at each level and the number of levels that must be visited.

Using an approach similar to Severence's[77], let z be the branching factor for a balanced tree containing N equally referenced leaves. For each retrieval the number of levels visited is:

$$L = \log_z N$$

and the average number of comparisons at each level is

$$C = (b+1)/2$$

to minimize the total number of comparisons

$$\min_{z\ 1} L * X = \min_{z\ 1} \frac{z+1}{2} * \log_z N$$

re-expressing the log of the base z to a base e and finding the minimum by taking the derivative and equating it to 0:

$$\frac{d}{dz}\ (L*C) = -\frac{b+1}{2} * \frac{\ln N}{z*(\ln z)^2} + \frac{\ln N}{\ln z} * \frac{1}{2} = 0$$

$$\frac{z+1}{z} = \ln z$$

$$z = 3.6$$

This analysis indicates the expected branching factor would be 3 or 4.

The questions of best search algorithms have been analyzed and addressed in literature [Donovan: 1972; Knuth: 1973; (see also bibliographies in these books]. Since storage is becoming so inexpensive and computer time critical, the trend is not to store elements in trees but rather to use hash searching techniques [Donovan: 1972]. This technique is accomplished by storing data in memory location whose value is computed by some function of the key.

Several techniques are commonly used when two keys results in the same value. One is a chained overflow where if a key is mapped into a location previously occupied, a new location is found and a pointer is

placed in the first location to it. Several analytical models have been developed to assist in determining the expected number of accesses required to retrieve a record. They all follow simialr formulations and result in the same. Using chaining and a bucket size of conclusions if a bucket size of 1 is used, the expected number is:

$$1 + P/2$$

where P is the loading factor. That is, $P = \frac{N}{M}$ where N equals number of records and M is the size of physical storage assuming a separate overflow area.

Analytical models in this area are based on probability theory.

8. Further Issues and Investigations in Hierarchies

The following issues are being or need to be investigated:

- Data about Data -- a quantitative theory for dealing with data of various degrees of confidence. Much of the data we have been dealing with is energy.
- Adaptive Restructuring -- develop a theory using the performance information on access tables to restructure data to make future access more efficient.
- Decision Rules for Efficient Queries -- at the relational level develop criteria that would determine under what conditions/a joint or a simple search should be performed. which operator should be involved, e.g., decide whether
- Multiple Access -- Highly active multiple users impose constraints of locking shared tables. The analysis of the implications of multiple access users at all levels will take place.

The following unproven theorems are proposed:

Theorem 1: Each element in the higherarchy has a well-defined set of operators. π_{ij} where i is the level and j differentiates between the operators.

Theorem 2: The set of operators at each level is equivalent to aximatic set theory. (Proof would formulate all the set theoretic operators in terms of the operators at that level.)

Theorem 3: If the operators π_{ij} are secure at a level below $i \leq k$, then any security violation must be at level n where $k < n \leq \max i$. (Proof would define security, show that because of the hierarchical implementation, namely, only calls inward are permitted. It is impossible to have security violation ripple down through secure levels.

Theorem 4: There exists a procedure for insuring system reliability of all systems implemented in a hierarchical fashion.

9. Further Investigations in Relations

Our future investigations at the relational level would include:

- Definition of new operators that can be implemented in micro-code.
- Develop algebraic techniques for reducing relational expression into some minimal set of operators.
- Formally prove the consistency and completeness of these new sets of operators.

Algebraic Techniques for Reduction:

The objective here is to find other operators that are more easily implemented, to show their equivalency, to further develop techniques for taking expressions of any relational operators, to reduce those expressions to equivalent expressions using only efficient operators, and finally, to reduce these expressions to minimal computational expressions. We propose the following theorems to accomplish the above objectives with proofs and blanks to be completed in the ensuing years.

Define a set of operators (π_1, π_2, π_3, π_4,...,π_j that)/are easy to implement on existing computers and have good perfomrance characteristics.

Theorem 1: The operators π_1, π_2, π_3,...,π_j are complete (proof could show their equivalence to operators of propositional calculus and hence are complete by Church's thesis).

Definition: Define a computational unit that is proportional to the length of time it takes a computer to execute a sequence of instructions.

Theorem 2: There exists a weighting function $f(\pi_i)$, that has a value for each π_i, and that value is proportional to a computational unit related to the performance of an implementation.

Theorem 3: For each sequence of operators T_j (where T_j is an unordered sequence of π_i) on a set of relations. There exists an equivalent sequence of operators P_k on the same set that results in the same relation. Proof would show an algorithm for reducing any string of operators (projections, restrictions, etc.) to an equivalent string of operators

Theorem 4: For T_j and P_k there exists the function

$$\sum_{\text{for all } \pi_i \text{ in } T_j} f(\pi_i) \quad \leq \quad \sum_{\text{for all } \pi \text{ in } P_k} f(\pi_k)$$

Theorem 5: It is always possible to determine the sequence k of P_k that is a minimum of the function of Theorem 4.

Theorems 1, 2, 3, 4, and 5 give formal insight into the question of equivalency of operators and sequence of operators that are equivalent and yet require less computation.

Theorem 6: If a relation has domains D_1, D_2, D_3,...D_n and if D_i is some function of D_j, then the following restrictions _____ make the updating problem soluble.

Thoerem 7: The set of operators π_1,...π_2 are consistent.

10. Comparison of Other Views of Data

The major attractiveness of the heirarchical model is that the model at each level is consistent, and simple in concept, which means routines of each level are always utilizing identical concepts, irrespective of the actual data. This also allows for extremely powerful storage/retrieval commands that do not inherently -- or otherwise -- contain a "path" to the data.

The hierarchical model affords many advantages (in fact, it incorporates) over conventioanl information system design. Within the context of the energy information system, this section will cite some of these advantages and compare them with other possible views of the data.

1) A system implementor can operate at a higher level than before, i.e., the data base need only be defined in terms of its relations and the operations upon those relations. This is in contrast to the conventional technique of first designing the internal file structure of the system followed by a large set of routines to manipulate that file structure.

2) Additional relations and additional domains within existing relations may be created after the initial implementation of an applications program without the need without the need to reprogram or reorganize the data base. For example, if it became necessary to start maintaining data on the distance of all fuel terminals from the water, and there was no previous provision for this do-

main in the energy information ssytem, a new domain called DISTANCE could be created in the TERMINALS relation without disturbing any of the existing data, and all future interrogations and manipulations on that domain would generate the correct answers for policymakers.

3) Since the rows of relations are order-independent, insertions and deletions to the data base can be handled with the same flexibility as described for additions of relations and domains.

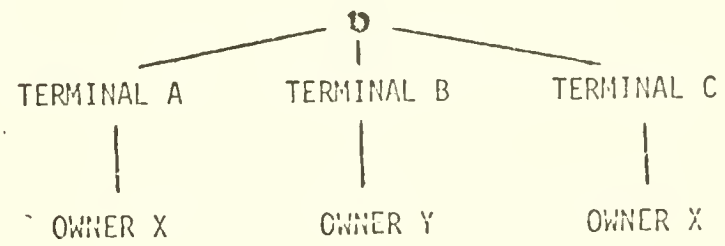
4) The use of system generated domain inversions provides an efficient and powerful retrieval capability that is much faster than a tuple by tuple linear search through the data base.

5) An attractive feature of the data security level is the fact that access control and integrity control are independent of the data structure, and so can be modified independently.

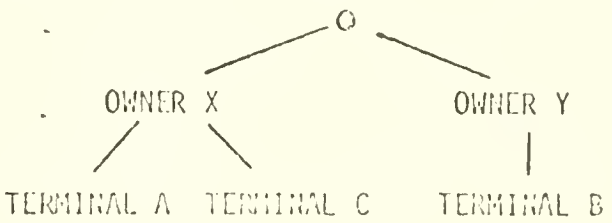
Other views of data tend to fail on one or more of the above ponts. If the energy system had been implemented in a non-hierarchical manner, say in FORTRAN, a fixed set of arrays would have to be defined and a maximum file size would be indicated at system generation time. If at any time in the future, any new field or domain was to be added to the data base, the entire system would have to be redone to handle the change. This means that the old compiled programs would have to be discarded.

A FORTRAN file would typically be sequentially ordered by some numeric key, such as terminal number for the terminals file. Any extensive insertions or deletions of terminal records would create a very inefficient data base and periodic manual data base reloads would be necessary. EVen the use of an indexed sequential access method (ISAM) would generate cumbersome overflow tables and pointers after many insertions and deletions to the data base, which could only be eliminated by complete reloading of the data. Interrogations would have to be satisfied through linear searches of the files, and perfomrance could be improved only through the implementation of specialized question-answering routines. New types of queries would require more progra-ming.

If the system was to be developed using a hierarchical tree view of th data, other problems would result. MUMPS [Barrett, Márble, Green, Papillardo of Mass General Hospital, Boston] is a high-level language with a powerful tree structure data base facility. A MUMPS version of an energy information system would have good update, deletion, ordering and insertion capabilities, but its ability to relate different types of information in the data base and to answer complex queries without extensive data redundance would be weak. As an example, consider the following hierarchy relating terminals to their owners:



A question to MUMPS asking for the owner of a terminal would easily be answered by accessing the next level of tree structure. However, if the question "What terminals does OWNER X own?" was asked, then the only way the answer could be found using the above structure would be to the owner of every terminal, and then collect all terminals owned by OWNER X. To improve performance, inversions are handled in MUMPS by using prelined bit maps of files, but that precludes extensive dynamic changes to the file structure. If the file was actually inverted like this:



then the data would have to be stored twice, and any changes that data would have to be made twice to maintain data integrity, not considering the waste of storage space.

11. Summary

The theoretical structures underlining our work are all related through the hierarchical concept. The relational model of data is simply the view of data at one level, a level above all physical dependencies. Each level has operations and performance issues that must be studied. The hierarchical concept itself is a structure that can be further studied and exploited.

The major thrust of the future research will be to:

- Define new operations at all levels
- Formally prove the following about the properties of operators at each level:
 - Completeness
 - Reduction to equivalent sequences of operators
 - Existence of analytical methods of determining minimal sequences
- Implement the following:
 - A modeling interface between this data management and a facility like TROLL
 - The NEEMIS application
 - Graphic interface
 - The complete DDL/DML -- security mechanisms and new operators (Levels 5, 6, 7, and 8)

- Formally prove the following characteristics of the hierarchical approach:
 - Under what conditions will there exist a deterministic procedure for proving correctness and integrity of any system constructed this way
 - The existence of a procedure for integrity and security of a hierarchical system
- Develop analytical models of performance at all levels
- Devise empirical performance experiments at all levels.

BIBLIOGRAPHY

Bracchi, G. et al.: "A Language for a Relational Data Base Management System," PROCEEDINGS, 5th PRINCETON CONFERENCE ON INFORMATION SCIENCE, 1972.

Bosak, .: "An InformationAlgebra," CACM, Arpil, 1962.

Chamberlin, .: IBM Research Laboratory, San Jose, 1974.

Church, ." "The Calculi of Lambda-Conversion," ANNALS OF MATHEMTICS STUDIES, No. 6, Princeton University Press, 1941.

Codd, .: "A Data Base Sublanguage Founded on the Relational Calculus," PROCEEDINGS, ACM SIGFIDET WORKSHOP, New York, 1971.

Donovan, J. & Jacoby, H.: "An Energy Information System for Aiding Public Policy Decisions in New England," PROCEEDINGS, NATIONAL ENERGY DATA WORKSHOP, Purdue University, October 1974.

Donovan, J. & Ledgard, H.: "A Formal System for the Specification of the Syntax and Translation of Computer Languages," PROCEEDINGS, FJCC, 1967.

Donovan, J.: SYSTEMS PROGRAMMING, McGraw-Hill, New York, 1972.

Donovan, J. & Madnick, S.: "Application and Analysis of the Virtual Machine Approach to Information System Security and Reliability," accepted for publication in IBM SYSTEMS JOURNAL, 1974.

DBTG CODASYL, ACM, New York, 1971.

Dijkstra, .: "T.H.E. Multiprogramming System," CACM, May, 1968.

Fehher, P.C.: "The Representation of Independent Language," IBM Technical Report RJ1121, November 1972.

Goldstein, I. & Strnad, A.: "The MACAIMS Data Management System", M.I.T. Project MAC TM-24, April 1971.

IBM IMS Manual H20-0524-1, IBM, White Plains, New York, 1968.

Hatfield, D.J.: "Some Experiments on the Relationship between Page Size and Program Access Patterns," IBM JOURNAL OF RESEARCH AND DEVELOPMENT, 16, Janaury 1972.

Loric, .: "XRAM", IBM Cambridge Scientific Center Report, 1972.

Madnick, S.E.: "Design Strategies for File System", M.I.T. Project MAC TR-78, October 1970.

Madnick, S.E. & Donovan, J.: OPERATING SYSTEMS, McGraw-Hill, New York, 1974.

Madnick, S.E.: "Storage Hierarchy Systems", Ph.D. Thesis, M.I.T., 1973.

Post, .: "Formal Reductions of the General Combinatorial Deicision Problem," AMERICAN JOURNAL OF MATHEMATICS, Vol. 65, 1943, pp. 197-215.

Quine, .: "New Foundations for Mathematical Logic, AMERICAN MATH MONTHLY, Vol. 44, 1938, pp. 70-80.

RAM, IBM Cambridge Scientific Center Report.

Seligman, Lawrence: "Experimental Data for the Working Set Model," M.I.T. Project MAC Computation Structures Group Memo No. 39, March 1968.

Severence, D.: "Use of Tree Structures," CACM, May 1963.

Smith, G.N.: M.I.T. Sloan School, ISG CISR Working Paper, November 1973.

Smullyan, .: "Theory of Formal Systems," Study 47, Princeton University Press, 1961.

3/21/75

APR 24 '78

DEC 05 '75

Date Due

SEP 21 '76

JAN 09 '80

FEB 25 '82

APR 26 '83

OCT 17 1991

JUL 0 6 1998

NOV 25 2000

OCT 31 1985

MAR 17 '88

OC 23 '89

Lib-26-67

MIT LIBRARIES
3 9080 003 702 773

BASEMENT

MIT LIBRARIES
3 9080 003 702 815

MIT LIBRARIES
3 9080 003 671 713

MIT LIBRARIES
3 9080 003 702 955

MIT LIBRARIES
3 9080 003 702 708

MIT LIBRARIES
3 9080 003 702 716

HD28.M414 no.759- 74
Holland, Danie/Tax aspects of alternat
731698 D*BKS 00037717
3 9080 000 866 795

HD28.M414 no.761- 75
Massachusetts /Industrial advertising
732083 D*BKS 00039462
3 9080 000 889 656

MIT LIBRARIES
3 9080 003 702 765